Probabilités Appliquées - Simulation de file d'attente

Léo Gourdin

Printemps 2018

Introduction

Le projet a été écrit en langage R. Le code n'est pas forcément le plus optimisé possible mais se veut le plus lisible possible.

Voici les conventions adoptées :

- *lambda* (λ) représente l'intensité des requêtes.
- *mu* (μ) représente le temps de service.
- Ces deux paramètres sont utilisés par une loi exponentielle grace à la fonction `rexp`.
- La variable (structure) t contient les éléments $t.clock$ et $t.end$ qui représentent respectivement le temps écoulé et le temps maximum virtuel de simulation.
- La variable q représente le nombre d'éléments dans la queue ET en traitement par le serveur (on suppose qu'une requête quitte la queue lorsque son traitement est terminé).
- Les variables $q1$, $q2$, et $q3$ représentent les parts de la queue q attribuées à chaque priorité. On a donc $q = q1 + q2 + q3$.
- Les variables $tours$, $nbLaunch/1/2/3$, $nbTerm/1/2/3$, $nbCancelled$ et $sommeReq$ servent à générer les indicateurs (compteurs, moyennes). Il est important de noter que les variables $nbLaunch/1/2/3$ et $nbTerm/1/2/3$ ne tiennent pas compte des requêtes annulées !
- La variable N représente le nombre maximum de requêtes dans le système, en comptant celle en cours de traitement. On a toujours $q \leq N$.
- Les paramètres $p1$, $p2$, et $p3$ représentent la proportion des requêtes prioritaires, normales et lentes, avec l'assertion de départ $p1 + p2 + p3 = 1$ (ligne 4).
- La deuxième assertion de départ est $\lambda \neq \mu$: On ne pourrait pas calculer le nombre moyen de reqêtes à un instant t théorique s'ils étaient égaux car on aurait $\rho = 1$ et donc une division par zéro dans la formule de l'espérance de X_t (ligne 5). Cependant, il est tout à fait possible de ne pas calculer cette espérance théorique et d'avoir les deux paramètres égaux à condition de commenter l'assertion et le calcul en question.
- La paramètre *debug* est un boolean qui permet d'afficher toutes les informations sur les requêtes durant l'exécution (lorsqu'il vaut TRUE).
- Le paramètre $plotXt$ est un boolean qui, lorsqu'il vaut TRUE, permet de calculer un graphique représentant X_t en fonction de $t.clock$. Attention, l'activation de ce paramètre ralentit énormément la simulation (il ne faut pas l'activer si le temps max est supérieur à 10000).
- On va traiter les éléments dans la queue selon le modèle fifo.
- Il est possible de n'utiliser qu'un seul type de requête avec le réglage suivant : $p1 = 1,\ p2 = 0,\ p3 = 0$.
- La simulation est dynamique : pas de structure de données complexes ou de génération statique au démarrage.
- La fonction `multiSim` a été conçue pour simuler plusieurs fois de suite en faisant varier uniquement la valeur de λ. Cette fonction va ensuite générer un graphique du pourcentage de requêtes perdues en fonction de λ.

Gestion des priorités

La priorité des requêtes est définie par le tirage d'un nombre aléatoire dans $[0, 1]$. Les requêtes sont ensuites placés dans la Queue. Pour cela, la variable q est incrémentée, et la variable q_x où x est le type de requeête est également incrémentée.

Le système va ensuite toujours traiter les requêtes par priorité, donc il commence par vérifier $q1$, puis $q2$, et enfin $q3$. Si jamais le système traite une requête non prioritaire et qu'une nouvelle requête, cette fois prioritaire, arrive, il va automatiquement la traiter une fois celle en cours terminée.

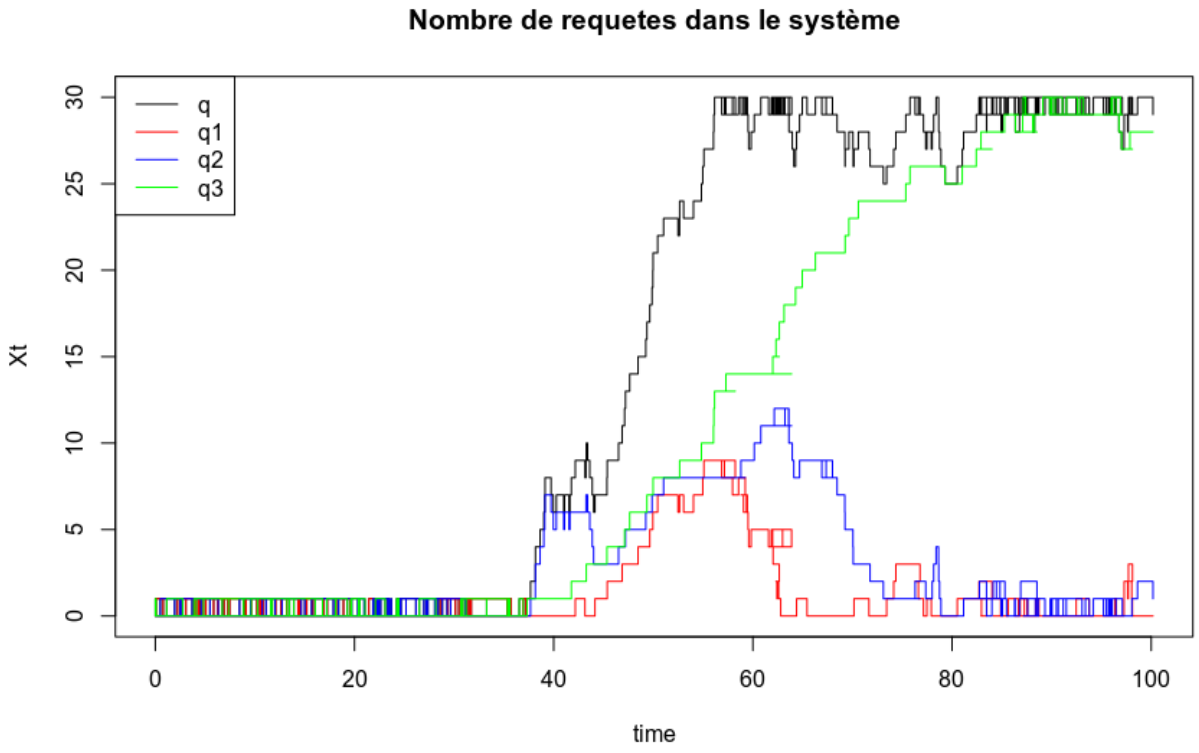
Nous venons donc en réalité de séparer notre queue en 3 "sous"-queues dynamiques de modèle fifo. La queue principale n'est donc plus dépendante du modèle fifo, puisque les priorités entrent en jeux. Ces trois "sous"-queues sont ensuites traitées dans l'ordre expliqué ci-dessus, avec pour contrainte de ne pas dépasser la taille maximale N à elles trois. Donc $(q = q1 + q2 + q3) \leq N$.

Il aurait aussi été possible de choisir un autre modèle de queue, comme un choix aléatoire. Mais en appliquant le système avec les trois "sous"-queues, ce choix n'aurait pas provoqué de changement dans le déroulé de la simulation (les jobs de même priorité uniquement auraient été traités dans un ordre aléatoire).

Nombre de reqêtes dans le système à un instant t

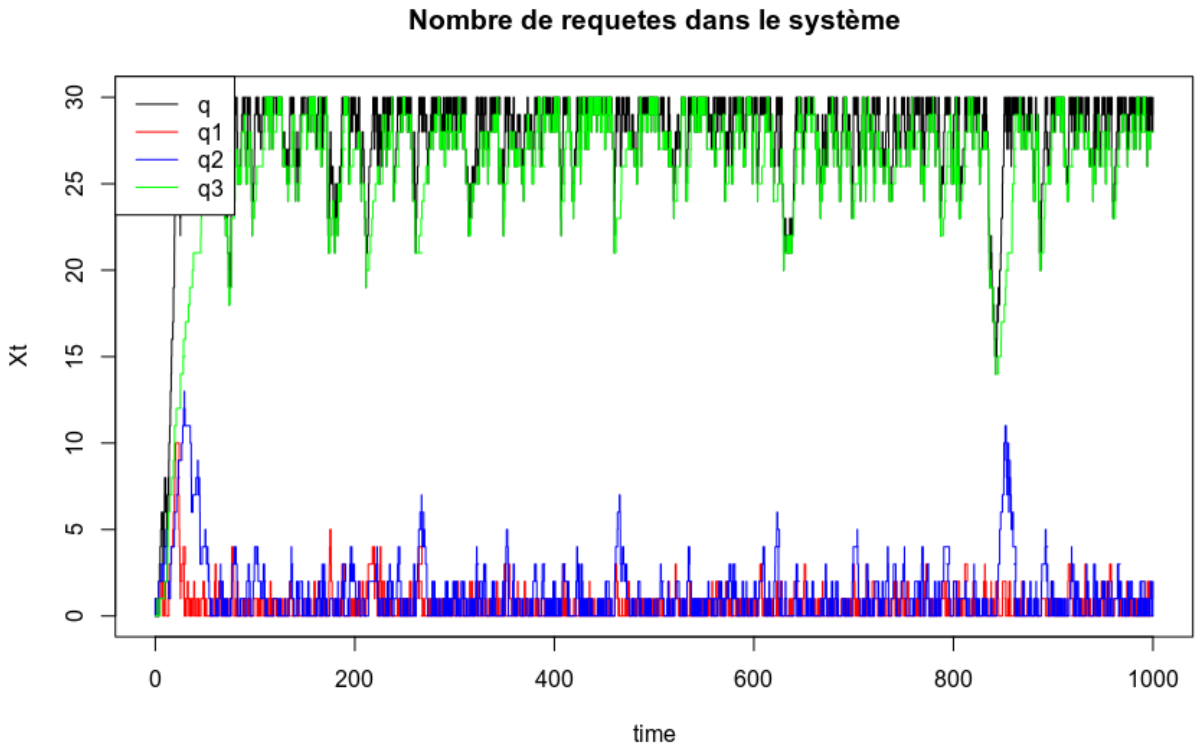
Le nombre de requêtes dans le système à un instant t est noté X_t. Dans ce script, ce nombre correspond à la variable q. Il est intéressant de représenter ce nombre sous forme de graphique pour voir, par exemple, si la queue est arrivée à saturation durant la simulation. Il est encore plus intéressant de représenter également $q1$, $q2$ et $q3$ sur le même graphique pour voir quel type de requêtes est le mieux traité par le serveur.

Voici un exemple de schéma, lancé avec la commande suivante `simQueue(2 ,1.9,30,100,1/3,1/3,1/3,FALSE,TRUE)`.



On remarque qu'au démarrage (jusqu'a environ un temps de 40), les requêtes sont traitées au fur et à mesure de leur génération, et donc $E(Xt) \simeq 1$. Ce phénomène se produit car lambda et mu on des valeurs très proches. L'intensité et le temps de traitement sont donc preque les mêmes.

Obsevons ce qu'il se passe sur une plus longue durée avec les mêmes paramètres.



On remarque que les requêtes ne s'accumulent pas dans les "sous"-queues 1 et 2 mais qu'elles sont très nombreuses dans la queue 3. En effet, comme le serveur traite en premier lieux les requêtes 1 et 2, il "laisse" plus facilement s'accumuler les requêtes 3.

Calcul et mesure du nombre moyen de requêtes dans le système

Le nombre moyen de requêtes dans le système est l'espérance de X_t, $E(X_t)$. Le script va calculer ce nombre théorique à chaque lancement, mais aussi mesurer le nombre moyen de requêtes dans le système par lui même. Cette mesure est réalisée grâce à la variable *sommeReq*.

Le nombre théorique et le nombre mesuré sont en général prochent sur un long temps de simulation.

Calcul et mesure du taux de perte

Le taux de perte, ou la probabilité de perte, représente le risque de perdre des requêtes suite à une surcharge de la queue. Ce taux est calculé de façon théorique par le script, et également mesuré. Les deux valeurs semblent se rapprocher lors d'une longue simulation.

Afin d'avoir des indicateurs sur les pertes, le script affiche également le pourcentage de requêtes perdues, et terminées (sans tenir compte des requêtes perdues pour le pourcentage de requêtes terminées, comme énoncé à l'introduction).

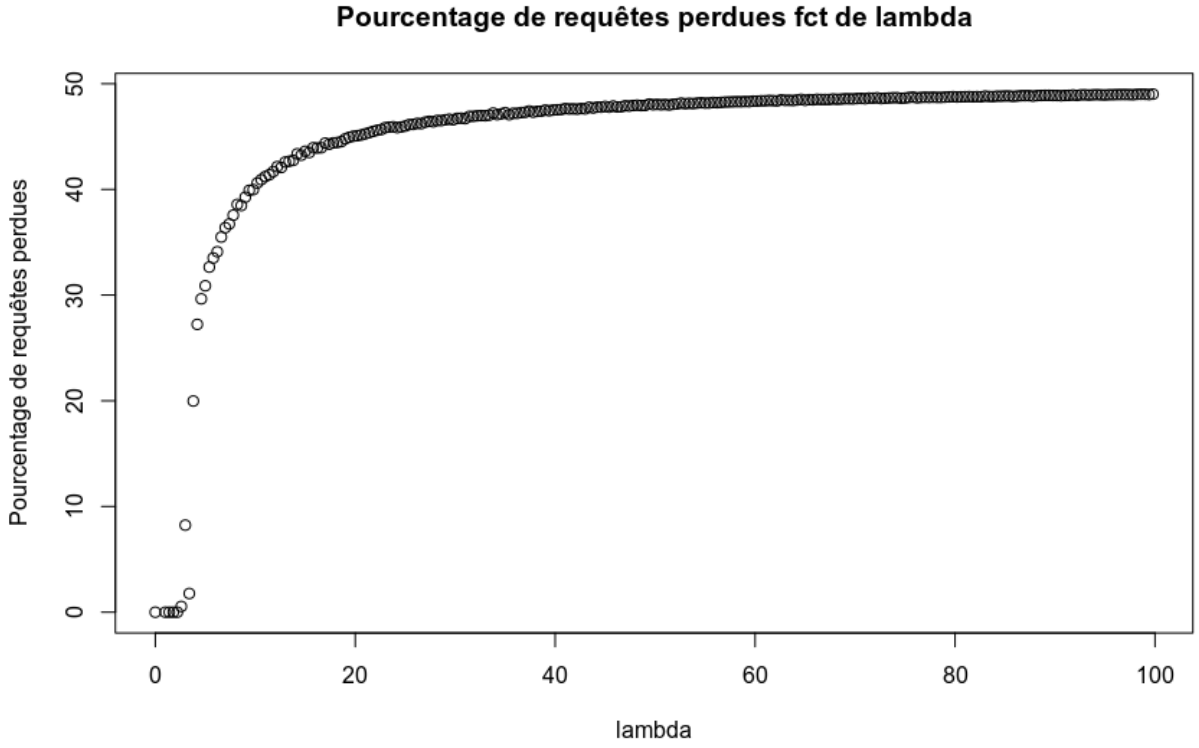
La probabilité de perte est la suivante :

$$P(X_t = N) = \frac{1-\rho}{1-\rho^{N+1}}\rho^N$$

D'une manière générale, on remarque que dès lors que $\lambda > \mu$, la probabilité de perte devient grande. Il serait donc cohérent de vouloir ajouter des serveurs si jamais le temps moyen de traitement est plus faible que l'intensité d'arrivé des requêtes.

Ici, on a appelé la fonction `multiSim` afin de lancer une série de simulations en faisant varier λ de 1 à 100 par pas de 0.4. Le pas a été choisi afin de ne pas se retrouver dans le cas $\lambda = \mu$. On a également fixé les valeurs de μ et N à respectivement 4 et 10. Vous pouvez observer la commande de simulation située dans la fonction `multiSim` : `simQueue(lambda ,4,10,1000,1/3,1/3,1/3,FALSE,FALSE)`.

Voici donc le graphique résultant du pourcentage de requêtes perdues par rapport à λ.



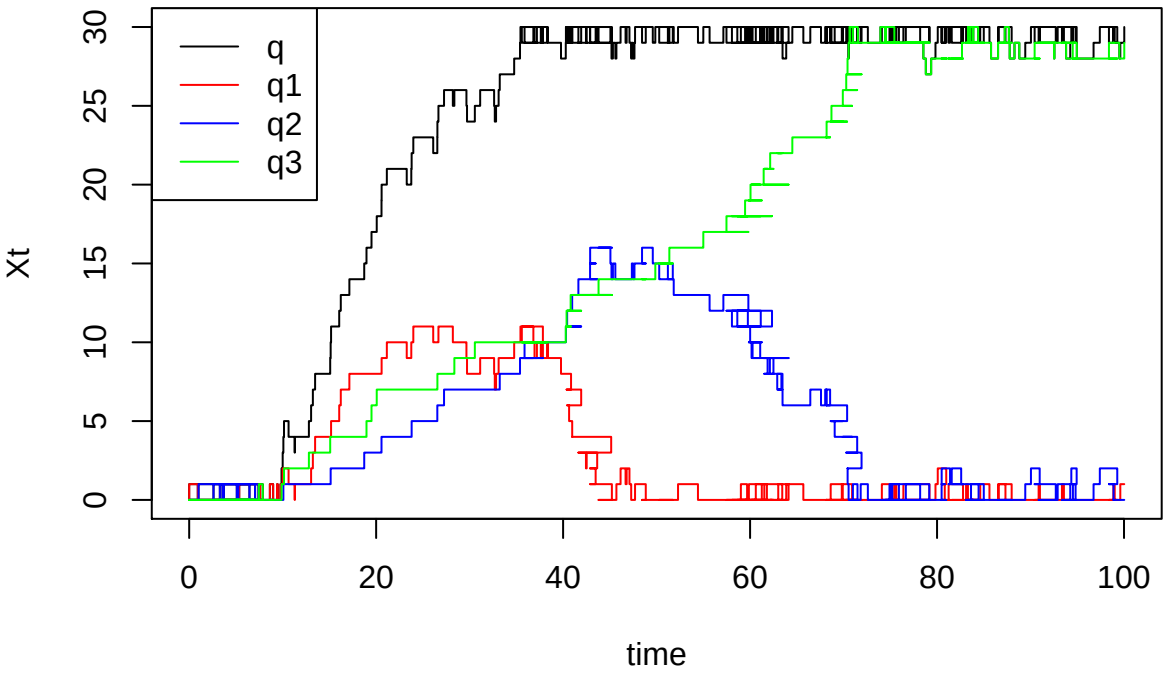
On peut observer sur ce graphe que, comme expliqué plus haut, dès lors que $\lambda > \mu$, le taux de perte augmente fortement. Cependant, ce taux semble ensuite progresser de façon logarithmique sans jamais dépasser les 50%. En effet, il y a toujours environ la moitié des requêtes qui pourront être traitées par le serveur.

Ce taux est un bon indicateur pour savoir s'il serait préférable ou non d'ajouter un serveur au système.

Un exemple qui fonctionne bien

On appelle la fonction `simQueue` de la façon suivante `simQueue(1.3333,1,30,100,1/3,1/3,1/3,FALSE,TRUE)`

```
## [1] "*****[INFO] Vérification des paramètres*****"
## [1] "*****[INFO] Initialisation du serveur*****"
## [1] "*****[INFO] Démarrage*****"
```




```
## [1] "*****[INFO] Nombre d'éléments en queue (total) :*****"
## [1] 30
## [1] "*****[INFO] Nombre d'éléments en queue (p1) :*****"
## [1] 1
## [1] "*****[INFO] Nombre d'éléments en queue (p2) :*****"
## [1] 0
## [1] "*****[INFO] Nombre d'éléments en queue (p3) :*****"
## [1] 29
## [1] "*****[INFO] Nombre de jobs lancés (total) :*****"
## [1] 155
## [1] "*****[INFO] Nombre de jobs lancés (p1) :*****"
## [1] 56
## [1] "*****[INFO] Nombre de jobs lancés (p2) :*****"
## [1] 52
## [1] "*****[INFO] Nombre de jobs lancés (p3) :*****"
## [1] 47
## [1] "*****[INFO] Nombre de jobs terminés (total) :*****"
## [1] 125
## [1] "*****[INFO] Nombre de jobs terminés (p1) :*****"
## [1] 55
## [1] "*****[INFO] Nombre de jobs terminés (p2) :*****"
## [1] 52
## [1] "*****[INFO] Nombre de jobs terminés (p3) :*****"
## [1] 18
## [1] "*****[INFO] Pourcentage de jobs terminés (total) :*****"
## [1] 80.64516
## [1] "*****[INFO] Pourcentage de jobs terminés (p1) :*****"
## [1] 98.21429
## [1] "*****[INFO] Pourcentage de jobs terminés (p2) :*****"
## [1] 100
```

```
## [1] "*****[INFO] Pourcentage de jobs terminés (p3) :*****"
## [1] 38.29787
## [1] "*****[INFO] Nombre de jobs annulés :*****"
## [1] 57
## [1] "*****[INFO] Pourcentage de jobs annulés :*****"
## [1] 26.88679
## [1] "*****[INFO] Nombre de tours :*****"
## [1] 280
## [1] "*****[INFO] Nombre moyen de requetes (simulé) :*****"
## [1] 25.17143
## [1] "*****[INFO] Nombre moyen de requetes (théorique) :*****"
## [1] 27.75398
## [1] "*****[INFO] Taux de perte (simulé) :*****"
## [1] 0.2688679
## [1] "*****[INFO] Taux de perte (théorique) :*****"
## [1] 0.2500148
## [1] "*****[INFO] Simulation terminée*****"
```

```
## [1] 26.88679
```